**1. Kafka Basics and Architecture**

* **What is Kafka and why is it used?**
* **Explain the core components of Kafka (Producer, Consumer, Broker, Topic, Partition).**
* **How does Kafka ensure message durability and fault tolerance?**

Kafka ensures **message durability** and **fault tolerance** using several key mechanisms:

**1. Replication**

* Each Kafka topic is divided into **partitions**, and each partition is replicated across multiple brokers. A partition has a **leader** replica and one or more **follower** replicas. The leader handles all read and write requests, while the followers replicate the data to ensure durability.
* If the leader fails, one of the followers automatically takes over as the new leader, ensuring high availability.

**2. Acknowledgement and ISR (In-Sync Replicas)**

* When a producer sends a message, it can choose how many acknowledgments it waits for before considering the message as successfully written:
  + acks=0: Producer doesn't wait for any acknowledgment.
  + acks=1: Producer waits for acknowledgment from the leader replica.
  + acks=all: Producer waits for acknowledgment from all **in-sync replicas (ISR)**.
* **ISR** is the set of replicas that are fully synchronized with the leader. A message is considered durable when it is replicated to all in-sync replicas (with acks=all), ensuring that data is not lost even if the leader fails.

**3. Data Persistence (Log Storage)**

* Kafka stores messages on disk in a **commit log**. Each partition has its own log file, and messages are written in an append-only fashion. This guarantees that once a message is written, it will not be lost due to crashes.
* Kafka uses **segment files** to store messages and indexes, and old data can be deleted or compacted according to configurable retention policies.

**4. Leader Election**

* Kafka uses **ZooKeeper** (or in more recent versions, Kafka’s own **KRaft** mode) to manage the cluster’s metadata and handle leader elections. When a broker fails, ZooKeeper coordinates the election of a new partition leader from the replicas, ensuring continuity without data loss.

**5. Producer Retries**

* Kafka producers can be configured with retry mechanisms to resend messages in case of transient errors or network issues, reducing the risk of message loss due to temporary failures.

**6. Data Checksumming**

* Kafka computes checksums for messages to ensure that the data written to disk is not corrupted. When a message is read, Kafka verifies the checksum to ensure data integrity.

**7. Consumer Offsets**

* Kafka stores consumer offsets in a special topic (\_\_consumer\_offsets) to track which messages have been consumed. This allows consumers to resume from where they left off in case of failure, ensuring no messages are missed or reprocessed unnecessarily.
* **What is a Kafka topic? How do partitions work in Kafka?**

**Kafka Topic**

A **Kafka topic** is a logical channel through which messages (records) are published and consumed in Kafka. It acts as a category or feed name to which records are sent by **producers** and from which **consumers** read messages. Topics organize the stream of data in Kafka.

* **Messages**: Kafka topics store streams of records or messages, and each message is associated with a key, value, and timestamp.
* **Decoupling**: Producers can write data to topics without concern about who is consuming it, and consumers can read from topics independently, enabling a decoupled, scalable architecture.

**Key Characteristics of Topics**

* **Log Structure**: Topics store messages in an ordered, immutable sequence of records. Once data is written to a topic, it remains available for a configurable period of time or indefinitely (depending on retention policies).
* **Multiple Subscribers**: Kafka topics support multiple consumers that can subscribe to the same topic and read the data at their own pace.
* **Durability**: Messages in a topic are durably stored, depending on the configured replication factor, ensuring high availability even in case of failures.

**Kafka Partitions**

Each Kafka topic is divided into **partitions**, which are the basic unit of parallelism and scalability in Kafka.

**How Partitions Work:**

1. **Parallelism & Scalability**:
   * Partitions enable Kafka to scale horizontally by distributing the data of a single topic across multiple brokers (servers).
   * Each partition can be stored on different brokers, allowing Kafka to handle large volumes of data and serve multiple consumers efficiently.
2. **Message Ordering**:
   * Messages within a single partition are ordered, and consumers read them sequentially. However, Kafka does **not guarantee ordering across partitions**. Therefore, if a topic has multiple partitions, message order across the topic is not guaranteed.
   * If strict ordering is required, a topic with a single partition can be used, but this limits scalability.
3. **Replication**:
   * Partitions are also the unit of **replication** in Kafka. Each partition has a **leader** replica and one or more **follower** replicas. The leader handles all reads and writes, while followers replicate the data for fault tolerance.
4. **Producer Keying**:
   * Producers can send messages to specific partitions based on message keys. Kafka uses a partitioning strategy (e.g., hashing the key) to determine which partition the message is written to.
   * If a key is not provided, Kafka will distribute messages across partitions using a round-robin or other strategies, but if the key is present, messages with the same key will always go to the same partition, preserving the order for that key.

**Partitioning Benefits:**

* **High Throughput**: By splitting a topic into multiple partitions, Kafka allows multiple producers and consumers to operate in parallel, improving throughput.
* **Fault Tolerance**: Kafka replicates each partition across brokers to prevent data loss if a broker fails.
* **Consumer Grouping**: Consumers in the same consumer group can read from different partitions of a topic, balancing the load across multiple consumers and enabling horizontal scaling.
* **What is Kafka’s role in a distributed system?**
* **Explain how Kafka guarantees message ordering.**
* **What is a Kafka Consumer Group and how does it work?**
* **How does Kafka handle message offset management?**

**2. Spring Boot Kafka Integration**

* **How do you integrate Kafka with Spring Boot?**
  + Can you explain the configuration required for Kafka Producer and Consumer in Spring Boot?
* **How would you send and receive messages in a Spring Boot application using Kafka?**
* **What is the role of @KafkaListener in Spring Boot Kafka?**
* **How do you configure topics, partitions, and replication factor in Spring Boot?**
* **What is KafkaTemplate in Spring Boot and how is it used?**
* **Can you explain how Spring Boot’s @EnableKafka annotation works?**
* **How would you configure multiple Kafka consumers with different concurrency levels in Spring Boot?**
* **How do you implement error handling in Kafka consumers using Spring Boot?**

**3. Advanced Kafka Topics**

* **How do you achieve exactly-once semantics in Kafka?**
  + Explain how idempotency and transactions work in Kafka.
* **What are Kafka Streams and how do they differ from the basic Kafka Producer/Consumer model?**
* **How does Kafka handle log compaction?**
* **What is Kafka’s ISR (In-Sync Replicas) and how does it help in fault tolerance?**
* **What is the difference between acks=all and acks=1 in Kafka producer configurations?**
* **How does Kafka manage backpressure with consumers?**
* **What is the significance of consumer lag, and how do you monitor it?**
* **How would you handle high message throughput in Kafka consumers?**
* **What is a dead letter queue (DLQ) and how do you configure it in Kafka using Spring Boot?**

**4. Message Handling and Processing**

* **How would you handle retries and dead-letter queues in Kafka consumer applications?**
* **What are the different acknowledgment modes in Kafka (MANUAL, AUTO, NONE)? How do they impact message processing?**
* **How would you implement a Kafka message filtering mechanism in Spring Boot?**
* **Explain the importance of message key in Kafka. How does Kafka use the message key to determine partitioning?**
* **How would you consume messages from multiple topics in Spring Boot using Kafka?**
* **How do you manage schema evolution in Kafka?**
  + Have you used Apache Avro or Protobuf with Kafka?

**5. Kafka Performance Tuning**

* **How would you optimize Kafka Producer performance?**
  + What are some important producer configuration settings that affect performance (e.g., batch size, linger.ms, buffer.memory)?
* **What factors affect Kafka Consumer performance, and how would you optimize it?**
  + How would you tune consumer configurations like max.poll.records, fetch.min.bytes, session.timeout.ms, etc.?
* **How do you handle large message payloads in Kafka?**
* **What is Kafka’s compression.type configuration? How does it improve performance?**
* **How does Kafka handle replication, and how would you optimize it in a multi-node cluster?**
* **How do you monitor Kafka’s performance metrics in production?**

**6. Kafka Security**

* **How would you configure SSL encryption between Kafka brokers and clients in Spring Boot?**
* **How do you secure communication between Kafka producers and consumers using SASL authentication?**
* **Explain Kafka’s role-based access control (RBAC) using Kafka ACLs. How do you configure ACLs for different roles?**

**7. Kafka Error Handling and Monitoring**

* **How do you implement retry mechanisms for failed messages in Kafka consumers?**
* **How do you handle deserialization errors in Kafka?**
  + Have you used Spring’s ErrorHandler to manage Kafka consumer errors?
* **What is Kafka’s consumer rebalance, and how do you handle rebalance events in Spring Boot?**
* **How would you monitor Kafka topics, consumer lag, and broker health?**
  + Have you used Kafka monitoring tools like Confluent Control Center, Prometheus, or Grafana?

**8. Real-World Kafka Scenarios**

* **What are some common challenges when integrating Kafka with Spring Boot in a production environment?**
* **Have you worked on any Kafka-based microservices architecture? Can you explain your experience?**
* **How do you ensure data consistency across different microservices when using Kafka as an event bus?**
* **How would you handle a scenario where a Kafka consumer application goes down and misses messages during downtime?**
* **How do you scale Kafka consumers in a Spring Boot application?**
* **What are the considerations for designing a Kafka-based event-driven system?**

**9. Kafka Transactions**

* **What are Kafka transactions, and how do they work?**
* **How would you implement a transactional Kafka producer and consumer in Spring Boot?**
* **What is the importance of isolation.level in Kafka transactions?**

**10. Kafka Use Cases**

* **Can you describe a Kafka use case that you worked on?**
* **How did you ensure high availability and fault tolerance in your Kafka setup?**
* **What steps did you take to handle Kafka consumer lags in a real-world project?**